Ready, Fire, Aim!

First things first: In my July 2006 BACKTALK article, "e-Dorado: The Lost Centric City of Information," the Office of the Secretary of Defense (OSD) net-centricity quote and reference were incorrect. The word *censors* should have been *sensors* and the reference should have been the OSD Net-Centric Checklist [1]. Ironically, only one astute reader caught the faux pas. In the wake of the Patriot Act and the National Security Agency's warrantless surveillance controversy, can I presume most of you thought censorship was part of net-centric warfare?

With quality on my mind, let's move on to this issue's theme – Software Assurance. The DoD relates software assurance *to the level of confidence that software functions as intended and is free of vulnerabilities* ... [2]. NASA adds that software assurance is *a planned and systematic set of activities that ensures that software processes and products conform to requirements, standards, and procedures* [3]. The essence of software assurance is predictability, trustworthiness, and conformance. Trustworthiness and process conformance receive plenty of press, so I will leave them to the security and process enthusiasts and focus on predictability.

With regard to predictability, the DoD wants software that is accurate (functions as intended) and precise (with confidence). I know, many readers are saying, "Gary, accuracy and precision are synonymous." Well, maybe in English 101, but not in science, engineering, and, as it turns out, marksmanship.

Several years ago, my son, Matthew, decided to obtain his rifle merit badge. After a safety lecture, we proceeded to the gun range where shooters sit side-by-side, taking 10 shots at individual targets similar to the target in Figure 1. Matthew was somewhat timid, having never handled a gun, so I agreed to shoot with him to ease his doubts.

We methodically fired 10 rounds, waited for the range to clear, and then marched to our targets to check our results. Concerned about my son's tally I watched as his shoulders slumped and his head dropped. This was not a good sign. I looked over his shoulder. His target looked exactly like Figure 1 – clean, no bullet holes, not even one! While considering how to handle the situation, I glanced at my target, which looked like Figure 2, and there was my answer.

Instead of 10 bullet holes, my target had 20. Matthew shot at my target and not his own. I shared my discovery with Matthew and his shoulders went back, his head snapped up, and later that day he earned the merit badge.

Although I could not differentiate my bullet holes from my son's, I did find the target pattern intriguing. Two distinct patterns appeared on the target. Ten holes were grouped around the center bull's-eye and slightly high. The other 10 holes are tightly grouped below and left of the center bull's-eye. In subsequent rounds, we found Matthew's targets to be tightly grouped and below center-left. I deduced that Matthew's shots were precise (his shots produced similar results – tightly grouped holes), but not accurate (his shots were below and left of the center bull's-eye) and my shots were more accurate (around the center bull's-eye) but less precise (not as tightly grouped).

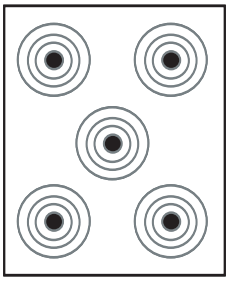

Figure 1: *Individual Target*

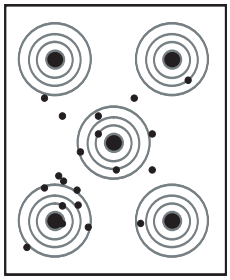

Figure 2: *Twenty Bullet Holes*

Figure 3 is a statistical view of accuracy versus precision. In software development terms, the "Actual Target" in Figure 3 represents the customer's bona fide requirements. The statistical depiction illustrates that you can be precise but not accurate, accurate but not precise or both. The DoD wants both – emulating user bona fide requirements (accuracy) with little variance over time (precision).

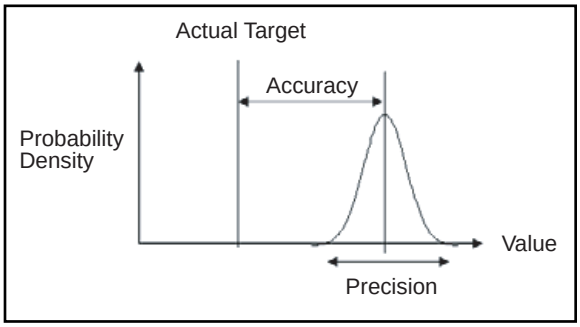

Figure 3: *Accuracy Versus Precision*

Accuracy requires that developers discern true reference values (requirements). However, the capricious nature of software requirements makes that discernment difficult, requiring focus, patience, and clear communication.

That reminds me of the story of the project manager who was late for a design review. Driving to the review, he tried to save time by slowing down and rolling through stop signs. A police officer stopped the project manager and cited him for failure to stop at a stop sign. Being a project manager, he naturally felt he was right and tried to convince the officer that he did slow down at each light. The officer replied, "But you did not stop." The project manager continued his plea, explaining that he slowed down and looked both ways. The officer replied, "But you did not stop." He explained that he slowed down and no one was hurt. The officer replied, "But you did not stop."

Rather than cutting his losses, the project manager, in typical project manager style, pressed the officer to the limit to avoid the ticket. Finally frustrated, the officer pulled the project manager from the car and started beating him with his baton. The project manager started to scream and yelled to the officer, "…stop, please stop!" To which the officer replied, "Do you want me to stop or just slow down?"

What's my point? Aim at the right target then worry about precision, beat your project manager occasionally, and know when to stop.

— Gary A. Petersen
Shim Enterprises, Inc.
gary.petersen@shiminc.com

References

1. Office of the Assistant Secretary of Defense for Networks and Information Integration/Department of Defense Chief Information Officer. "Net-Centric Checklist 2.1.3." 12 May, 2004.
2. Department of Defense Software Assurance Initiative. 13 Sept. 2005.
3. National Aeronautics and Space Administration. "Software Assurance Standard." NASA-STD-2201-93. 10 Nov. 2002.